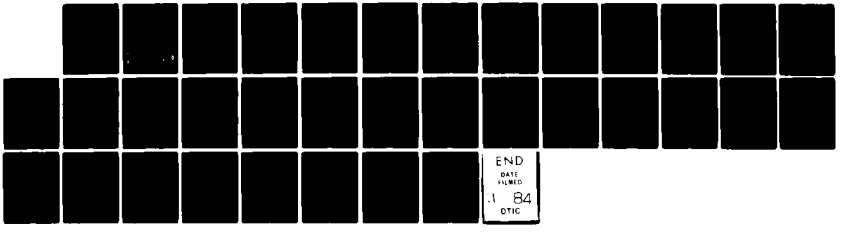
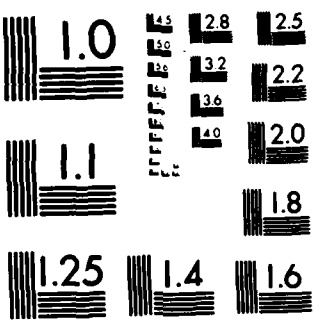
AD-A136 523 DESIGN OF OFFICE INFORMATION SYSTEMS(U) UNIVERSITY OF 1/1
SOUTHERN CALIFORNIA LOS ANGELES DEPT OF COMPUTER
SCIENCE E HOROWITZ ET AL. 08 NOV 83 AFOSR-TR-83-1253
UNCLASSIFIED AFOSR-82-0232 F/G 5/1 NL





UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

(2)

AD A136523

DTIC FILE COPY

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS:
2a. SECURITY CLASSIFICATION AUTHORITY:	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE:	
4. PERFORMING ORGANIZATION REPORT NUMBER(S):	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 83-1253
6a. NAME OF PERFORMING ORGANIZATION: University of Southern California; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code): Computer Science Department, University Park, Los Angeles CA 90089-0782	7b. ADDRESS (City, State and ZIP Code): Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-82-0232
8c. ADDRESS (City, State and ZIP Code): Bolling AFB DC 20332	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A2	

11. TITLE (Include Security Classification): DESIGN OF OFFICE INFORMATION SYSTEMS

12. PERSONAL AUTHOR(S): Ellis Horowitz and Balaji Narasimhan

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1983	27

16. SUPPLEMENTARY NOTATION:

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Office Information System (OIS) design is currently an active field of research. The authors outline the essential components of a truly integrated OIS. Then they critically examine four of the existing prototype systems and another suggested design. These systems have the common characteristic of providing a form-based user interface. Then they present a set of requirements for such an OIS.

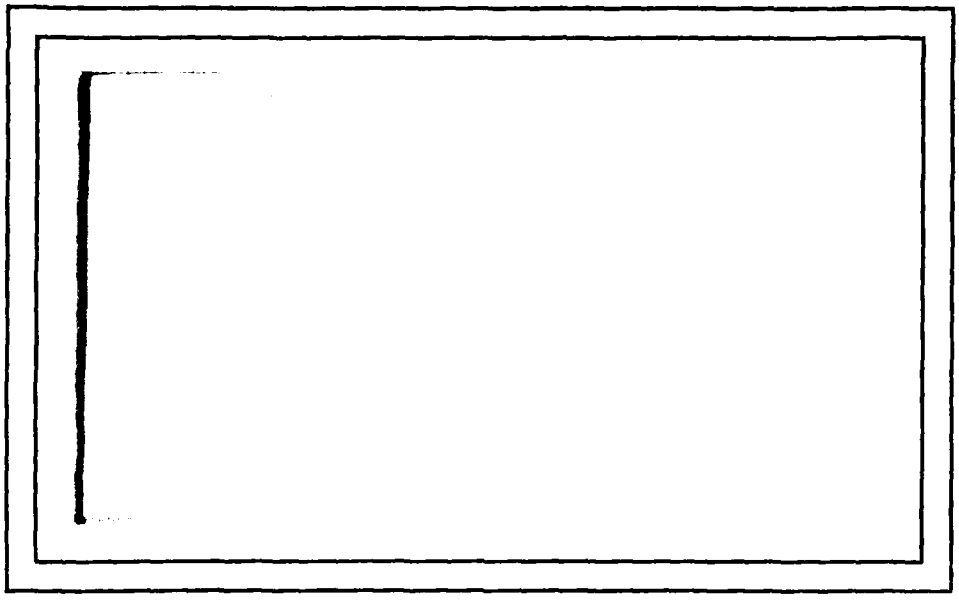
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Dr. Robert N. Buchal	(202) 767- 4939	NM

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AFOSR-TR- 83-1253



COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90089-0782

Design of Office Information Systems

Ellis Horowitz
Balaji Narasimhan

Design of Office Information Systems

Ellis Horowitz and Balaji Narasimhan
Computer Science Department
University of Southern California
Los Angeles, Ca. 90089

8 November 1983

This work was supported in part by the Air Force Office of Scientific Research Grant no. AFOSR 82-0232

Table of Contents

List of Figures

Design of Office Information Systems

Abstract

Office Information System(OIS) design is currently an active field of research. We outline the essential components of a truly integrated OIS. Then we critically examine four of the existing prototype systems and another suggested design. These systems have the common characteristic of providing a form-based user interface. Then we present a set of requirements for such an OIS.

1. Introduction

An office is a place where an organization manages the information about its operations. It gathers information from the outside world, manipulates the information, creates new information, and disseminates it both within the office and to the outside world. In order to do these tasks, an organization employs people. The typical tools employed by the people in performing these tasks are: typewriters, paper, memos, folders, files, filing cabinets, calculators, telephones, dictaphones, mailboxes, postal and other similar mail services. A recent study [2] estimated the investment in office equipment to be about $3000 per offfice worker, as against a capital investment of $50000 for each farm worker, and $35000 for each factory worker. The reason for this situation has been that office workers manipulate "information"; and until the last eight to ten years the high cost of information processing tools - computers - has far outweighed any gains to be made by their use in the office. However, technological advances in microelectronics has made the cost of computers, computer peripherals, and computer communication drop significantly, often at rates of over 20% per year [18]. At the same time, the cost of human labor has been increasing at 6 to 12% per year [2, 18]. This rise in cost has also been paralleled by the increasing information handling requirements of offices. The combination of these factors has stimulated research into the use of computers as tools to support, enhance, and/or replace the everyday activities of office workers so as to reduce the total cost of running an office [4, 15]. This study is called *Office Automation* research.

In section 2 of this paper, we introduce the basic components of an Office Information System (or OIS) and discuss the difference between a collection of tools and a truly integrated office system. In section 3, we examine some of the existing prototype OISs and designs. A common characteristic of these systems is that they are oriented towards documents which have a visual structure to reflect the semantics of their contents. We call such documents *forms*. Then in section 4 of the paper, we state the requirements

for an OIS based on forms.

2. Office Information Systems

In this section we describe the nature of the work performed in an office and show how present computer systems attempt to support the various activites. Then we introduce the notion of an office information system as an office environment in which these various activities are supported by a model of office activities and by the use of forms for the user interface with the system.

In order to understand the scope of the use of computers in the office, let us first look more closely at the types of activities performed there. Office activities may be divided into the following four classes.

Document Preparation: This refers to the preparation of letters, memoranda, invoices, customer notifications, technical manuals, financial reports, business proposals, and so forth. The documents usually contain text, but may also contain diagrams, illustrations, charts, and other graphical elements. The creation of a document may also require that some computations be performed. The contents of a document may be forced to follow a predefined format, such as in an invoice; this we call a form. Or, it may be relatively free from such constraints. Documents are prepared by secretaries, professionals, or managers and the skills of a draftsman are sometimes employed. The tools typically used in document preparation are typewriters, drafting tools, and calculators. When a document has to be made in quantities, a photocopier is used.

Communication: This refers to exchange of information both within the office, and between the office and the outside world. Both text and voice are used as media of communication by offices. Text communication is supported by telex, postal and other similar mail services, couriers, and manual delivery of messages within an office room. Voice communication is supported by face-to-face conversations, telephones, recorded messages, dictaphones, etc. Communication may be between one person and another, or between one person and a group.

Information Storage/Retrieval: Information is stored in the form of documents in files, folders, and filing cabinets, or in the form of entries in ledgers, and logbooks. Knowledge of how the documents are filed, or the entries made is made use of in retrieving information from these information storage media.

Personal support activities: This refers to the maintenance of calendars and scheduling

of appointments or travel. These activities are performed either by the person concerned, or by a secretary responsible to that person.

Assuming today's level of computer technology, there exists a great deal of software for supporting each of the types of office activities. There are word processors to help in document preparation. There is electronic mail to facilitate communication. There are database systems for information storage and retrieval and there are software packages automating such tasks as maintaining a personal calendar. However, these tools are not ideal for use in the office for the following two reasons.

First, activities in an office are not performed in isolation from one another. For example a secretary in an academic department might retrieve some student information from a file, and use that information in preparing a summary report about the newly admitted students; once such a report has been prepared, he will have to send the report to a professor; and the secretary might want to be reminded by his calendar about the due date for the report. Such a task involves, information retrieval, document preparation, communication, and the use of a personal support tool. This is true, in general, about all activities performed in an office. Office work involves performing activities of these various types concomitantly; and information created or obtained by one type of activity may be used as a component of the information handled by another activity. Software used to support office activities should directly support the interdependent nature of these activities.

Secondly, the user interfaces of the present day office software tools are non-uniform. For example, the user commands or interactions meaningful in the word processor may bear little or no relation to commands or interactions meaningful in a database system. This situation is due to the fact that these office software tools were conceived of and designed independently of one another. However, it makes these tools difficult to learn and become familiar with for the unsophisticated users of these systems in the office. And having to switch contexts mentally in going from one tool to another can be a constant source of annoyance to the user.

For these two reasons, we claim that we need to integrate the software tools supporting the different types of office activities into a single software system; such a system should employ single paradigm to provide a uniform user interface. We say that such a system provides an *integrated environment* for performing office activities.

2.1. Office Models

So far we only talked about activities which are performed directly by the office workers. Some of the current office automation research looks a little beyond that and attempts to study office work as information processes which go on in the work place. The researchers have suggested models to represent office work. Examples of such models are Augmented Petri Networks (APN) [17], Information Control Nets [4], and BPDL [12].

A formal representation of office work can serve many useful purposes.

1. It can clarify the office tasks to the people who perform them.

2. It can bring out the inconsistencies, and inefficiencies in the ways the office work is currently performed.

3. For the above reason, it can be used to reorganize office work in meaningful ways.

4. A formal specification can be used by a computer to monitor the performance of work, and keep track of the flow of information in the office.

5. A formal specification can be used by a computer to carry out activities which do not require human participation; thus, a computer could become an active participant in the office, instead of being merely a passive tool.

3. Examples of OIS

In this section we present examples of systems which have been proposed as office tools. We consider four existing prototype OIS: STAR [9, 13], SCOOP [17, 18], OFS/TLA [16], and Odyssey [6]. We also consider FPL/BPDL, a design suggested by N. C. Shu, et al. of IBM [12]. We examine how each of these systems supports office activities, the integration of the different tools for these purposes, and the models, if any, used to represent office work in these systems. We do not provide detailed descriptions of these different systems; for such descriptions, we refer the reader to the references cited above.

3.1. STAR

Star is a powerful personal computer developed at Xerox Palo Alto Research Center in 1981 [9, 13]. It provides tools for document preparation, communication using electronic mail, and information storage/retrieval functions. The interface to these tools is unified in terms of paradigms of operations on graphical objects, as we explain later. The designers have chosen not to consider the problem of modelling office work.

3.1.1. Star User Interface

The Star interface is centered around a high resolution graphic display and a pointing device called the mouse. The mouse may be moved on any flat surface such as a table; this movement is sensed and used to guide a cursor on the display.

Star has been designed for use by professionals or as its dsigners call them knowledge workers such as financial analysts, research and development scientists and engineers, technical writers, and so forth. These people are expected to spend less time on large routine tasks (unlike clerical personnel); they need fast and comprehensive processing, quality graphic and typographic facilities, and effective communication capabilities. However skilled these users are in their own professions, they are not expected to have computer sophistication. Therefore, the designers have tried to provide a model to the users that will be familiar and uniform throughout the system.

The user's initial view of the Star display is a desktop which shows pictorial representations or icons of the familiar objects in the office: documents, folders, filing cabinets, in- and out-baskets, printers, calculators, and so forth. Any icon may be selected by moving the cursor over that icon and pressing one of the two buttons provided on the mouse. Once an icon has been selected it can be moved about on the 'desktop', copied, or removed from the 'desktop' by pressing the appropriate command key: MOVE, COPY, or DELETE. When a destination must be specified, it is done using the mouse. An icon which has contents, such as a document or folder may be opened by first selecting that icon and pressing the OPEN key. This causes the contents of the icon to be displayed in a rectangular area called a window. In addition to operations such as MOVE, COPY, and DELETE, other operations are provided in a menu. To perform an operation from a menu, the user 'selects' it using the mouse.

3.1.2. Documents in Star

Star allows the user to create virtually all types of documents on the display. The user can exercise complete control over the visual appearance of the form using the graphics primitives, and the choice of fonts for text. Letters appropriate to several Europian languages, non-Roman scripts, and special symbols for office applications, math and logic are also supported by Star. The contents of a document may be manipulated in the same way as one manipulates the icons, namely, by selecting with the help of the mouse, and using the appropriate command keyMOVE, COPY, or DELETE. One can print a document by selecting it or an icon representing it and moving it to the icon representing the printer. One can mail a document by moving the icon to the out-basket and indicating the recipient's address by selecting an icon that represents the recipient.

In the language of the designers of STAR, the icons, windows, documents are all known as objects. Objects in STAR have properties appropriate to their types. For example, a document has the properties: name, size, owner, and create/read/write time stamps. The user can access the properties of an object by first selecting it and then pressing the command key PROP. A new window is opened on the screen to display the properties of that object. When necessary and appropriate one can change a property in the usual way using the mouse. For example, one can rename a document by first deleting the old name from its property sheet and then typing in the new name.

A Star document can be a form such as an invoice etc. A form has one or more fields which can be thought of as containers for values. A field is treated as an object; one can edit the property sheet of a field to define the name, type, format, range, and optionally a fill in rule. A fill in rule is an expression specified in a language called CUSP (CUStomer Programming language). CUSP is a simple language with no side effects, and no control constructs except the conditional expression. It provides operators for arithmetic and string concatenation, and aggregate operations like SUM and COUNT, comparison and Boolean operators. The designers of Star expect to make CUSP a full programming language eventually, by supporting procedures, variables, parameters, and a programming environment.

3.1.3. Information Storage/Retrieval

Records Processing is the Star facility for information storage and retireval. It refers to a set of operations for handling collections of forms. A collection of instances of forms of the same type is called a record file. To create a new record file one first selects an icon called empty record file from the 'desktop'. This causes a new window to be displayed.

The user selects a form which has the field structure desired for the new record file, and invokes a command called Define Structure from the menu of the window. This is in contrast to systems such as SQL [3] which require the use of a data definition language to define the field structure of a record file.

The correspondence between the field structure of record files and of documents carries over into all other accesses to record file data: addition, display, deletion, and modification of records. Multiple forms may be associated with a record file to provide varied forms of display of the same data. Each such document is called a display document.

A display document may contain only a subset of the fields in a record. It may, however, contain additional fields with fill in rules to compute aggregate functions. Its format may be that of a tabular report with data from many records gathered into a single document; or its format may be that of a form each instance of which corresponds to a single record.

When a record file is displayed as tabular report, one can add, delete, or modify records by the addition, deletion, or modification of the rows of the tabular report. When a record file is displayed through forms one record at a time, one performs addition, and deletion of a record using menu commands. One can update a record by first updating the display document and then confirming the changes using the menu. Whenever a record is updated or a new record added to a record file, validation is performed according to the property sheets of the fields.

A user queries a record file by a process known as filtering. A filter is a predicate on the fields of the record file. It defines a subset of records that the user is interested in. The records which satisfy the predicates are displayed to the user as a table. A filter is defined simply by defining the property sheet of this table - in particular by specifying desired ranges for its fields.

A view of a Star record file consists of a sort order, a view filter, and a display form. A view can be thought of as the encapsulation of one distinct use of a record file. Each distinct use of a record file may require that a view be created to support it. The sort order for a view specifies the order in which the records in that view appear; an index is maintained internally for each sort order. A view filter functions as a permanent query on the record file; whenever the record file is accessed using a view, only those records which pass the view filter are displayed to the user. A display form is any Star document;

normally the field structure of a display form corresponds to that of the record file.

A record file may be manipulated at the icon level in Star. It can be printed by moving the icon to the printer icon. New records may be inserted by moving or copying a document to the record file icon. A record file is transferred to a file server by moving or copying its icon to a file drawer icon on the 'desktop'. A record file is mailed by moving it to an out-basket icon. When a record file is moved or copied, only those records which passed the view filter during the last access to that file are affected.

3.1.4. Discussion

Star represents significant progress in the design of user interfaces. The use of the mouse combined with the high definition graphics lets the user accomplish many things without using the keyboard. The designers have consistently tried to make the operations using the mouse have uniform semantics throughout the system. The uniformity extends to all other aspects of the system. This should go a long way in making the system easy to learn and use.

The designers of Star claim they did not intend their system for use on routine, high volume tasks that one associates with clerical personnel. Hence, the problem of office procedure specification is not addressed at all.

The query facility of Star is attractive and powerful. However, it is not so powerful as QBE [19] to which it is similar. All the queries one can express in Star are necessarily limited to data on one record file. There is no way one can correlate the data from two or more record files.

The absence at this time of a full programming language in Star may prove a weakness when it becomes necessary to program using Star. However, the designers hope to alleviate this by making CUSP a full-fledged programming language.

3.2. SCOOP

SCOOP is a system developed by Michael Zisman and others at the Wharton School, University of Pennsylvania, in the year 1977 [17]. It is a system for modelling office work, and for monitoring the performance of office work using the models.

The formalism which SCOOP provides for modelling office work is called Augmented Petri Nets (APN). APNs are based on Petri nets [8], a formalism for describing asynchronous processes, and on production rules [7], a knowledge representation

technique used in artificial intelligence.

A Petri net consists of two types of nodes, called places and transitions, and directed arcs each of which interconnects a place and a transistion (or a transition and a place). It is usual to depict a Petri net pictorially, with circles denoting places, dark lines denoting transitions, and thin lines showing the directed arcs. A place can be either empty or can contain a token. When a place contains a token, the transition(s) on the outgoing arcs from that place are enabled to fire. (We omit finer details, such as multiple incoming arcs at a transition, from this brief description. We refer the reader to [8] for such details.) When a transition fires, the place(s) on the outgoing arcs from the enabled transtion get new tokens.

A production rule consists of a boolean condition and a set of actions to be taken if that condition evaluates to 'true'.

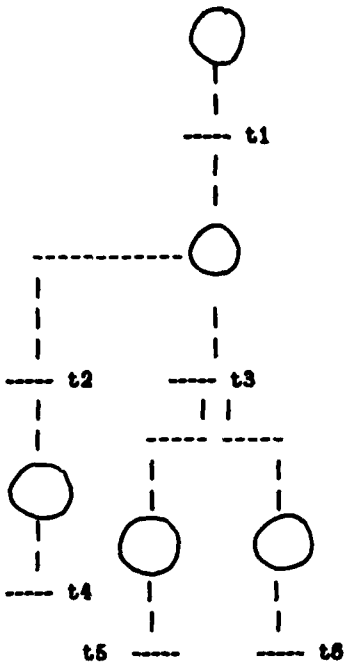
An APN consists of a Petri net with sets of production rules attached to each transition in the net. Each transition in the net corresponds to an office activity. The production rules describe the activities.

Let us consider an example to show the use of an APN to define a common office function - processing orders in the distribution center of a company. The Petri net and the associated production rules are shown in figure 3-1. When an order is received from a customer it is first logged in(t1). The inventory record for that item is checked to see if sufficient quantity of the item which has been ordered is available. If not, a letter of apology is generated(t2) and mailed(t4) to the customer. (In a more realistic situation orders would be placed with the manufacturer to procure more of that item.) Otherwise, the order is forwarded to the shipping and billing sections(t3). The shipping department ships the item from the stocks to the customer(t5). An invoice is also made out and mailed to the customer(t6).

The important points to note from this example are:

- How Petri nets can be used to depict the asynchronous, concurrent nature of office work. Thus, activities at transitions t5 and t6, shipping the ordered item and mailing the invoice can go on in parallel.

- How decision making is captured by the production rules. Thus, depending on whether inventory is less than or is at least equal to the quantity ordered by the customer, an apology is mailed, or actions are taken to ship the item (transitions t2 and t3). Only one of the two transitions will eventually 'fire'.



Production Rules:

t1: log order
check inventory level

t2: if (inventory < ordered quantity) produce apology letter

t3: if (inventory >= ordered quantity) forward order

t4: mail apology

t5: ship ordered item

t6: mail invoice

Figure 3-1: Augmented Petri Net description of Processing Orders

This example shows a description of one function performed in an office. Corresponding to each function of an office, there will be an APN describing it. It is a plan of how to carry out that function. Its APN description is given to the execution monitor of SCOOP when it is desired to carry out the function. If the same function is to be carried out several times concurrently, several instances of the APN can be created and given to the execution monitor.

The execution monitor maintains a data structure for each instance of an APN. This data structure shows which places have tokens, which transitions are enabled, and the values of the variables used in the production rules of the APN. For each enabled transtion, the monitor tries to execute the production rules attached to that transition by evaluating the boolean condition, and performing the actions which follow the condition if the condition is true. A transition is considered to have been fired if the boolean conditions of all the production rules evaluate to true, and all the actions specified in the rules are executed.

3.2.1. Discussion

The main contribution of SCOOP was that it was the first system in which the concept of a formal model of office work was brought out clearly. As pointed out in [17] it was necessary to shift the emphasis from automating separate tasks to automating office functions. APN formalism is a notation to represent how office functions are carried out in an office.

The view taken of office procedures in SCOOP is very rigid. It assumes that one could define accurately and with certainty how office functions are to be carried out. However, this is not always true. An evidence of this is the derogatory connotations of the word "bureaucracy". Circumstances not anticipated by the designer of a system will always arise in the course of the use of an OIS [5]. In such circumstances, the OIS should positively aid the user in dealing with the new situation; or in the least, it should not obstruct the user from dealing with the new situation.

One of the features not considered in the design of SCOOP is the integration of information storage/retrieval activities into the system. As the designer of SCOOP himself pointed out, this would bring closer together the automated office and the data processing function.

In addition to the formalism of APNs, Zisman proposed a non-procedural language interface to SCOOP. Though the language itself was neither formalized nor implemented, it raised what is in our view an important issue: A higher level formalism for modeling office work than APNs. This remains an important issue in the design of computer systems for the office.

3.3. OFS/TLA

OFS and TLA are two systems developed at the University of Toronto. These two together with a relational database management system called MRS can be used for performing document preparation, communication, and information storage/retrieval activities in office. TLA can be used to model the activities of any single individual in an office.

Documents which can be most naturally defined and used in this system are forms similar to the ones we considered in STAR, though this system lacks the graphics capabilities of STAR. OFS is a system for defining form templates, and creating form instances. TLA is a system for defining automatic operations on forms. In defining form templates in OFS, one can assign properties to the fields of a form such as valid ranges of values, whether the value in a field can be modified after the form instance is created and whether the value of a field can be left undefined. OFS provides a simple editor for use when the form instances are created. It also provides operations to mail form instances, or to retrieve form instances from a local file. Form instances created using OFS can be accessed using a DBMS called MRS (Micro Relational System). MRS treats the values of the fields of a form as defining a tuple of a relation.

OFS is a passive system in the sense that it cannot be used to specify automatic operations. For example, in many business forms one field should be computed as a function of the values in other fields. OFS is not adequate for specifying such operations. TLA(Toronto Latest Acronym) is a system for specifying such operations and the conditions under which those operations are to be performed. One can also use TLA to relate the fields of different forms in a fashion similar to OBE [19]. The operations (conditions) are specified on templates of the forms on which they should be performed (checked). Such a specification is called a sketch.(Please see figures 3-2 and 3-3) If a specification involves data external to the form itself, such as where a form came from, one uses pseudoforms provided by TLA. Thus, if an operation should be performed only on forms originating from a particular person, one specifies that condition on a source pseudosketch. One can specify where a form should be mailed to using a destination pseudosketch.

A TLA procedure is defined by a set of sketches defining conditions and actions. A set of interrelated form instances which can be used by a TLA procedure is called a working set. Whenever a form instance is created, or arrives at a station by mail, the TLA interpreter checks if it belongs in a working set. Once a complete working set of forms

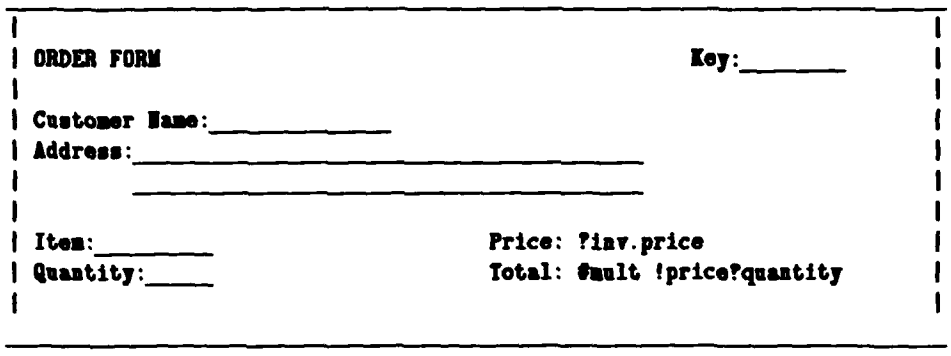


Figure 3-2: A Sketch Defining Automatic Operations in TLA

Notes: '?inv.price' causes the price to be obtained from the inventory form.

The 'Total' field is automatically computed by a function call to 'mult' with 'price' and 'quantity' as arguments.

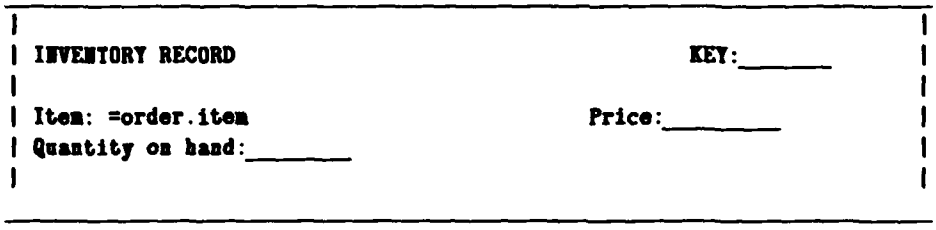


Figure 3-3: Precondition Sketch

Note: This sketch specifies the inventory record which has an 'item' name matching the name in the order

of a TLA procedure is thus assembled, it performs the operations specified in the form sketches.

3.3.1. Discussion

In contrast to SCOOP, this system starts from an external model of office activities, i.e. form operations and proceeds to express office work in terms of form operations. Though it is not clear that one can specify all office workin this way, we believe forms are a good paradigm as user interfaces.

The notion of a working set in TLA permits one to relate operations on different forms. However, it allows one to relate only forms pertinent to one workstation. We consider the TLA approach to defining office work by defining form operations valuable. However, such definitions are all local to the workstation at which they are to operate. There is no global view of the operation of the office that can be modeled using TLA.

The automatic form operations one can perform using OFS are rather limited. Though one can specify automatic operations using TLA this is not a satisfactory solution. In general, the TLA interpreter waits for a complete working set of forms of a procedure to arrive before it performs an automatic operation, and once it performs the operation, all the forms of the working set are removed from the set of available forms. Thus TLA would not be useful if an automatic operation depended upon, say, a local database of forms received at a station.

When a form instance is accessed using the database management system MRS, the constraints specified in the form template are not maintained. One can change the value in any field of the form instance using MRS. As pointed out by Tsichritzis in [16], the reason for this situation is historical. MRS was a relational DBMS already available when OFS was implemented, and hence it was used to access the files created using OFS. Ideally, one would have liked OFS to be compatible with the database management system.

The relationship between OFS and MRS is not symmetric. Though one can access the data on the form instances using MRS, one cannot use OFS and TLA to access the database. OFS and TLA only operate on forms that are created, modified or received at a station. This is not a satisfactory situation as, very often, one obtains the data which go into a form from a database.

Since OFS/TLA is to be used in a distributed environment with numerous workstations, a distributed query capability should also be provided. But the only way one can handle distributed queries in OFS/TLA is by the elaborate process of sending messages containing the query to the workstations and waiting for the responses.

3.4. Odyssey

Odyssey is an information system specialized to a particular office application, namely, planning of trips for individuals in an office. It is not an integrated system for supporting office activities or a system for modeling office work. However, we have

included it in this paper in view of its use of artificial intelligence techniques in an office system.

Odyssey was designed and implemented by Richard Fikes and Warren Teitelman [6]at Xerox Parc in 1980. It was part of an experiment to study how an information system could use knowledge about an office enviroment to assist a user in performing office tasks. The two key features of this system are knowledge representation and a form based user interface. We will present an overview of the system and examine these features of the system in this section.

3.4.1. System Overview

Odyssey comprises a form system, a knowledge-base, and a set of databases. The form system is based on Officetalk-Zero [4]. It provides a form-based user interface; all interactions of the user with the system are operations on forms displayed on a video screen - specifically, the user interacts by filling in fields of forms, updating old values in the forms, or deleting fields of forms. The knowledge-base contains information about objects in the system. Examples of objects are forms, fields of forms, and users. The information concerning each object can be of two types: what kinds of values it can have, and procedural information, i.e., how to perform certain operations on the objects. The databases provide descriptive information which is made use of by the knowledge-base procedures. Examples of these databases are profiles on individuals, and properties of cities such as the airports in a city.

The system presents itself to the user as a set of predefined forms. The system starts by displaying a menu with an entry for each step in a trip preparation - plan travel between cities, plan lodging for overnight stays, request reservations, plan activities for each day of the trip, and request advance money. When a user selects an entry from the menu, the system displays a form appropriate to that step in planning the trip. The information entered into these forms is collected into a database. The system assists the user in filling out the forms by utilizing its knowledge of the task involved. This knowledge deals with the structure of trips, the steps involved in trip preparation, properties of dates and times, cities, airports, and personal profiles of individuals. The system also maintains areas for repeated fields of the form, adding or deleting areas as needed. Whenever possible, it fills in fields using data retrieved from other forms or databases.

3.4.2. Knowledge Representation in Odyssey

Odyssey was implemented in KRL1, an interlisp implementation of KRL which is a knowledge representation language developed by Daniel Bobrow and Terry Winograd [1]. In order to understand how knowledge representation is done in Odyssey, we need to explain some aspects of KRL1.

Conceptually a system programmed in KRL1 comprises a set of frames. Each frame is a specification of an abstract data type. It consists of a collection of message types to which an instance of the data type can respond, and a collection of slot descriptions. Slots are analogous to local variables in programming languages with the difference that each slot is also assigned an abstract data type. A slot can respond to messages specified in its own description, in addition to those specified in the declaration of its data type.

The frames in the system can bear a subclass-superclass relation to one another. A frame X which is a subclass of a frame Y inherits the procedural information from Y. This characteristic enables the users of KRL1 to define a hierarchy of datatypes, without repeating identical information in the definition of the datatypes in the hierarchy.

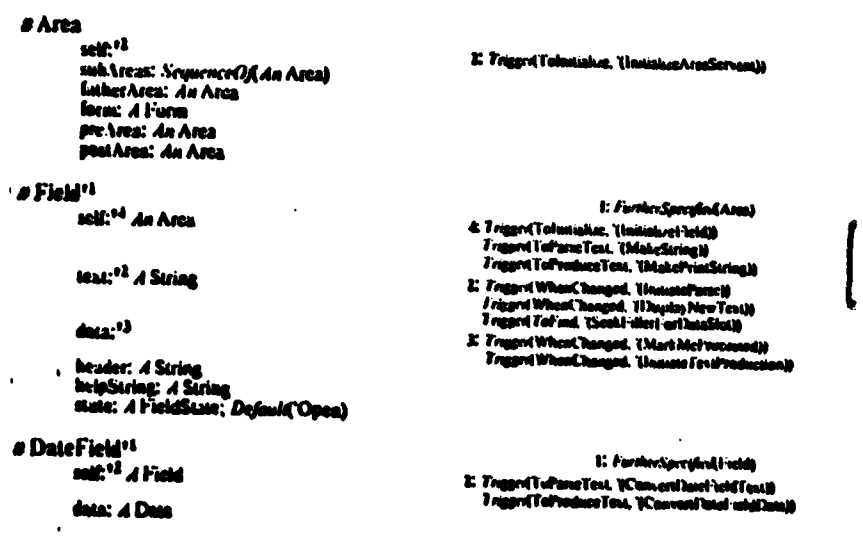


Figure 3-4: Examples of Data Types in KRL1

Figure 3-4 shows examples of data type declarations in KRL1. It consists of declarations of three data types - Area, Field, DateField. Area is a superclass of Field which in turn is a superclass of DateField. The type Area consists of a procedure 'InitializeAreaServant' and slots subAreas, fatherArea, form, preArea, and postArea. The

procedure 'InitializeAreaServant' is itself written in lisp and not shown in the frame. It is evaluated whenever the message 'ToInitialize' is sent to an instance of the type 'Area'.

Slot descriptions are mechanisms to represent knowledge in the system. Five types of properties of slots can be specified in KRL1.

1. linkages between equivalent slots

2. default equivalence linkages

3. default values

4. frame actions to compute descriptions

5. frame actions to compute values

Equivalent slots are those which must have identical values at all times. The specification of linkages permits the system to obtain the values of equivalent values to be obtained automatically whenever one of those slots has been assigned a value. Default equivalence linkages are similar, except that the user may override the equivalence by explicitly assigning distinct values to the slots. Frame actions are used to perform actions which cannot be directly stated in KRL1. Such actions, coded in lisp, are invoked as procedures from KRL1. The messages which are usually used to invoke these procedures are WhenChanged and ToFind. WhenChanged procedures are used when the value of a slot to which the procedure is attached is changed. Such a procedure can be used to cause values for other slots to be computed or cause any other needed change to the data base. ToFind procedures are used to compute default values by using other values in the database. Whenever a value required by such a procedure is not present (probable reasons could be that the user did not fill in a field of a form or had not created the required form instance), the ToFind procedure would wait for the value to be available.

In Odyssey, fields of forms are made to correspond to slots of the types above. Therefore, slot descriptions are used directly to perform form operations. The slot descriptions are used by the runtime system to assist the user in doing these operations and maintaining the consistency of the data on the forms. Thus, if the values of fields A and B are used to compute the value of a field C, then system would perform this

computation automatically when both A and B have values. The fields may be filled (assigned values) in any order. Further, if the values of A or B are changed, the system would recompute the value of C automatically. If the values of these fields are further used in other places in the system, the system can propagate the effect of the changes in the values of A, B, and C. If the user attempts to modify the value of C, the system would inform the user of the resulting inconsistency.

3.4.3. Discussion

The most noteworthy aspect of this system is the introduction of an AI perspective on office information systems. One could argue that the results achieved by Odyssey might well be achieved by using other systems. The KRL1 representations of information could even be viewed as "mere programs". However, the possibility would still remain that we could exploit the experiences and techniques of workers in artificial intelligence to organize and use effectively information in the office domain.

Embedding knowledge of the office domain permits many operations to be performed automatically. Previously, one used to consider an "electronic spreadsheet" to be an intelligent form; such a form could automatically compute one field from other fields of the same form using user-supplied expressions, compute subtotals, averages, and so forth. Now the "intelligence" of a form could extend to the entire office domain. A form could exhibit knowledge about the existence and properties of other forms, and databases in the system. It could interact with the user more gracefully. For example, Odyssey allows the user a liberal set of input formats, and performs error correction whenever possible on the inputs. Moreover, the use of the knowledge-base also extends to the changes in the database. Whenever the user makes changes by updating some information on a form the system makes other necessary changes implied by it on other forms. Corresponding changes are also made in the underlying database.

Odyssey is not a system for general office use as it is specialized to a particular task. It does not provide an integrated environment for the specification and performance of office functions. Nor is it easy to program the system to perform new tasks. To do so would require knowledge of KRL1, officetalk, and lisp besides the details of implementation of Odyssey itself.

KRL1 itself, the language in which most of Odyssey has been programmed, was originally designed to understand human mental processes involved in understanding natural languages. While this should not preclude other uses for it, it might be useful to investigate other techniques suitable for representing office domain information in an

OIS, without bringing in the anthropomorphic considerations which went into the design of KRL1. One of the features of such a technique could be a notation for procedural information (which is represented in KRL1 using Lisp).

One issue not considered in the design of Odyssey itself, but which presents itself in many artificial intelligence problems is "learning" (by the system). A system which is able to learn will be able to modify its knowledge-base either gradually or at times chosen by the users. For example, when an unforeseen situation arises during the performance of an office procedure, the system could try to remember how the user responds. The knowledge of this response could be used later when a similar situation arises. The advantages of such a system are twofold. It will be able to tailor itself to the user depending on the user's pattern of use. It will be able to evolve more gracefully to accommodate changes in the structures of the office task and the office.

3.5. FPL/BPDL

This is a system which has been proposed by Shu et al. of IBM. They define a forms processing language (FPL) for specifying the operations on a form. Form processes specified in this way are integrated using a non-procedural language, Business Process Definition Language(BPDL), to model office work. One can use this sytem for preparing documents (forms), for communication using forms, and for information storage/retrieval activities. One can model office work using BPDL in terms of preparation of documents and communication using them.

FPL provides facilities for specifying the characteristics of the values in the fields of a form, such as their type, uniqueness within a form instance, range constraints, whether any field could be left undefined at the time the form instance is created, and how copies of the form instances should be created; in this category one can specify different types of copies in each of which a different subset of the fields of the original are visible. These characteristics have to be specified once for each type of form. Instances of the form may be created or updated differently at different times. This is called a form process specification. Thus, one can specify how the values that go into the fields are to be obtained, such as whether they have to be entered manually, or from some other forms, items to be matched when constructing an output form instance from one or more input form instances, and how to select input form instances for processing. Each process is given a name as part of its definition.

BPDL provides commands for performing form processes which have been defined using

FPL, and for stating the conditions for doing so. In addition it provides a command for routing a form instance to one or more destinations which can be a user, a list of users, a workstation, another form process and so on. A program in BPDL consists of a sequence of these commands; it reflects how a certain function is carried out in an office.

3.5.1. Discussion

The distinction made between a form type (or template) and a form process is an important one. This reflects the situation of form instances of a particular type being involved in different sets of operations. This distinction is not brought out explicitly in OFS/TLA.

Many of the processes that go on in an office are asynchronous. For this reason BPDL seems to be ill-suited to specifying them. If the individual activities that make up such a process are defined to be operations on a form, then it should be possible to relate those activities into a process by including enough information in the definitions of the form operations themselves. In OFS/TLA, for example, this is done by specifying relationships between the fields of distinct forms, and sketches on pseudoforms. Whereas there could be many more solutions to this problem, BPDL seems to be the least desirable one.

4. Requirements for an OIS

An OIS is an integrated system for the modeling and performance of office tasks. We can amplify this definition with a set of high level requirements for such a system. These requirements are given without a specific implementation in mind. Therefore it should be possible to evaluate an existing system against these as the criteria. Another use for these requirements could be that these can be used as a guide in designing a new system.

In this section we first give a set of high level requirements for an OIS. Then we consider the design of form-based OIS satisfying these requirements.

4.1. High Level Requirements for an OIS

1. An OIS should provide a user interface which
   a. is designed around a uniform paradigm; the user actions should have uniform semantics throughout the system.
   b. requires a minimum of input from the users, without compromising on possibile ambiguities.

c. gives meaningful and immediate feedback of the right amount to the user on his interactions.

An OIS should support a model of office work.

2. An OIS should assist the office workers in the execution of office procedures.

3. An OIS should support monitoring the progress of office work.

4. An OIS should provide management tools to oversee the running of the office.

4.2. A Form-based OIS

Our specification of the high level requirements for an OIS are sufficiently broad to allow varying types of specific designs. In this section we would like to consider the design of an OIS based on forms. We clarify this intent of ours first, and follow it with a set of functional requirements for such an OIS.

4.2.1. Forms as an OIS Paradigm

Forms have been employed to provide user interfaces in many systems. In addition to the systems we have considered in this survey, there are database systems [10, 11], application systems such as Screen Cobol [14] among others which emphasize the use of forms in this respect. Form-based interface is a goal of COUSIN, a project currently underway at Carnegie-Mellon University which is aimed towards providing a friendly interface between users and programmed systems, including operating systems and application programs.

The reason for this is mainly the availability of high resolution graphics displays and pointing devices such as the mouse which can be used with the displays. This provides an expanded bandwidth for communication between a computer and a user. (It is true that some of the systems we have referred to above have been implemented on line-oriented display devices, but that is more due to immediate practical considerations at the sites where the work was performed than due to a lack of awareness of the new technology.) Forms provide one way to use this larger bandwidth to capture in the act of communication more of the semantics of the message itself. For example, if the user wants to delete a record from a file, then the semantics of deleting a 'row' from a 'table' displayed on the screen is closer to his intent than the semantics a command or a

sequence of commands which would accomplish his intent. (In this example, the message is 'to delete a certain record from a certain file.)

Therefore, we would like to explore the use of forms in an OIS which would satisfy the high level requirements we have specified above. To that end we give the functional requirements for such a system here.

4.2.2. Requirements for a Form-based OIS

1. Definition of form templates: This is a well designed man-machine interface that permits a non-sophisticated computer end-user to create a form template. A form template refers to the external appearance of a form, and properties of and relationships between fields.

The external appearance of a form aids in bringing out the meaning of the contents of a form to the viewer. It may be considered under four aspects as follows.

1. Annotation: This aspect of the visual appearance of a form template refers to the text and pictures which must appear repeatedly in each form instance. The main function of this is to interpret the meaning of the fields of a form. It also serves the auxiliary function of being decorative in some cases.

2. Emphasis: This aspect guides the person viewing a form to the most significant fields first, and reduces the distraction due to the less important fields which must, nevertheless, be there.

3. Contrast: This aspect draws one's attention to the difference between some of the fields of a form when that difference is important to the viewer. For example, in a form that shows the revenues and expenditures of an organization it may be helpful to show the opposing nature of the fields.

4. Placement: This aspect aids one to infer the relationship between the fields of a form from their relative placement. For example the field showing the arithmetic sum of a set of numeric fields may be placed below those fields.

These aspects of a form template may be implemented by the use of different typefaces, fonts, sizes, and colors. The degree to which these aspects of a form can be controlled by the user depends on the hardware support available, and the software support for using the hardware.

The external appearance of a form is only one property of a form. It may have additional properties of type, range constraints, whether a field can be left undefined at the time a form instance is created, and optionally a rule to compute the value automatically for each form instance. Such a rule may involve references to data outside the form itself, for example other forms, or a database.

Designing a form template means defining its external appearance and the properties of its fields. The conventional approach to this issue is to embed a set of programming language constructs in a programming language to define form templates. This is the approach typified by database systems such as screen Rigel [11]. A drawback of this approach is its non-interactive nature; i.e. it does not provide immediate visual feedback to the user about the form template that is being designed. Since we expect the OIS to be used by people who neither know nor wish to learn a programming language this approach is not appropriate. The second approach, the one taken by the designers of STAR among others, is to provide an interactive graphics system for form definition. This does not have the drawbacks of the first approach.

2. Definition of actions on form instances: This is the definition of what gets done with the form instances once they are created. As distinct from the definitions made on a form template, the information relevant to these actions do not appear on for instances. This defines the role of instances of the given form template in the office task. The actions which must be supported by a form system are mailing a form instance, archiving it, printing or making copies of it.

To perform each of these actions, the user may need to specify additional information. Mailing a form instance requires that the name(s) and address(es) of the recipient(s) be specified. Archiving requires that a filename be specified. (It is possible that these pieces of information may be defined at the time a form template is defined, so that the user need not specify the same for each form instance.)

One must also be able to define under what conditions such actions are to be taken. These conditions can be either on time or on the values of the fields of a form instance or a database. A condition defined on time specifies at what time (the day, hour, minute, or

second) an action is to be taken. It can be either an absolute time (August 1st, 12 Noon, 1983) or time relative another event (Two weeks after mailing the invoice send a reminder).

3. Validation of form instance creation: This facility is required to support the creation of form instances according to the definitions of the corresponding form templates. Once again this should be interactive in order to allow the user to create form instances in the same way as one fills out paper form blanks.

There is a question of how interactive this facility should be made. For example, if the value a user enters into a form field violates the constraints specified on the form template, the user may be told so either immediately or after the user has entered all the possible information. The decision to do either way may not be trivial, as ther are advantages to both the alternatives.

4. Execution of actions on form instances: The OIS should support a facility to carry out actions on form instances according to the definitions described in requirement 2. Some of the actions could be fully automatic, such as mailing a form to a particular person. But sometimes it may be necessary to elicit some details about the actions from the user, such as a specification of the mailing destination of a form instance after it has been created. In such cases, again information should be obtained from the user through an interface of forms.

5. Manual intervention: The form templates and the actions one has defined should not limit what one can do with the office system. One must be able to perform ad hoc activities as the needs arise. Hence, the OIS should provide a facility to allow the user to override the actions described in requirement 2 and define new actions at form instance creation time; or the impelementation of requirement 2 should take this need into consideration.

6. Store and retrieve form instances: This is a facility to aid the user in managing the collection of form instances at his worksite, and utilize it, when necessary in other form activities. Thus it should enable the user to store form instances, retrieve form instances which satisfy some properties, and create reports, both tabular and graphical, from the data on the form instances.

7. Office procedure specification: When one defines a set of form templates and actions on form instances, one is in effect defining how an organization carries out a function in

an office. The OIS should support the use of form definitions as specifications of office procedures. In other words, it should be possible to specify office procedures through a set of form definitions. The OIS should use this specification to monitor the performance of each instance of the office procedure. The form activities are, then, component activities essential to the performance of an office function as specified in the office procedure.

References

[1] Bobrow, Daniel G. and Winograd, Terry.
An overview of KRL, a knowledge representation language.
Cognitive Science 1(1):3-46, 1977.

[2] News in Perspective.
Advice for Uncle Sam.
Datamation (1):67, Jan., 1981.

[3] Date, c.j.
The systems programming series. Volume 1: *An Introduction to Database Systems.*
Addison Wesley, Reading, MA, 1981.

[4] Ellis, Clarence A., and Nutt, Gary J.
Office Information Systems and Computer Science.
Computing Surveys 12(1):27-60, March, 1980.

[5] Fikes, Richard E. and Henderson, D. Austin, Jr.
On supporting the use of procedures in office work.
In *Proc. of the First Annual National Conference on AI*, pages 202-207. AAAI, The American Assoc. for Artificial Intelligence, Aug., 1980.

[6] Fikes, Richard E.
Odyssey: A knowledge-based assistant.
Artificial Intelligence 16:331-361, 1981.

[7] Newell, A., and Simon, H.
Human Problem Solving.
Prentice-Hall, Englewood Cliffs, N.J., 1972.

[8] Peterson, J.L.
Petri Nets.
ACM Comput. Surv. 9(3):223-252, Sept, 1977.

[9] Purvy, Robert, et al.
The Design of Star's Records Processing: Data Processing for the Noncomputer Professional.
ACM TOOIS 1(1):3-24, Jan., 1983.

[10] Rowe, Lawrence A., and Shoens, Kurt A.
A Form Application Development System.
In *???*, pages 28-38. ACM, 1982.

[11] Rowe L.A., et al.
Programming language constructs for screen definition.
IEEE Trans. Software Engineering SE-9(1):31-39, Jan., 1983.

[12] Shu, N. C., et al.
Specification of Forms Processing and Business Procedures for Office Automation.
IEEE Transactions on Software Engineering SE-8(5):499-512, September, 1982.

[13] Smith, David C., et al.
The Star user interface: an overview.
In AFIPS (editor), *Proc. AFIPS NCC*, pages 515-528. AFIPS, Arlington, Va., 1982.

[14] Tandem.
Tandem 16 Pathway Reference Manual.
Technical Report 82041, Tandem Computers Inc., Feb., 1980.

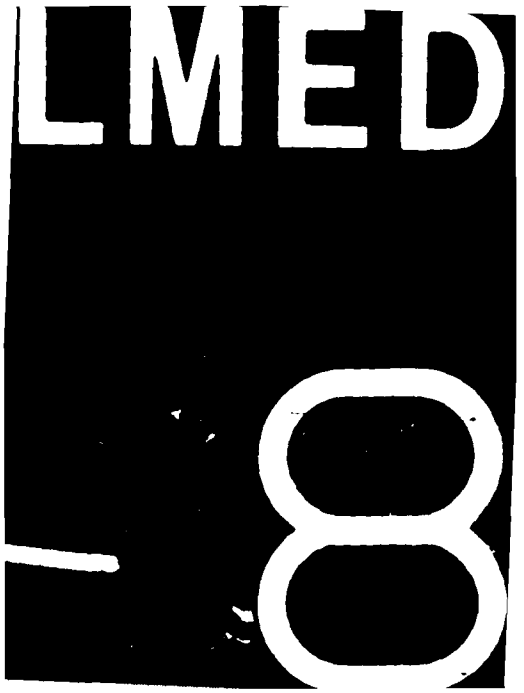
[15] Tsichritzis, D.C., and Lochovsky, F.H.
Office Information Systems: Challenges for the 80's.
Proc. IEEE 68(9):1054-1059, Sep., 1980.

[16] Tsichritzis, D.C.
Form Procedures.
CACM 25(7):453-478, July, 1982.

[17] Zisman, M.D.
Representation, Specification, and Automation of Office Procedures.
PhD thesis, Wharton School, University of Pennsylvania, 1977.

[18] Zisman, M. D.
Office Automation: Evolution or Revolution?
Sloane Management Review 19(3):1-16, Spring, 1978.

[19] Zloof, M.
QBE/OBE: A Language for Office and Business Automation.
IEEE Computer 14(5):13-22, May, 1981.


LMED
8